OSProj3 Multithreaded Sorting Application & Fork-Join Sorting Application

by 潘禧辰，518021910497

Menu

OSProj3 Multithreaded Sorting Application & Fork-Join Sorting Application
- Menu
- Abstract
- Environment
- Quick Start
  - 编译
    - gcc编译
    - jdk编译
  - 测试代码
    - Multithreaded Sorting Application
    - Fork-Join Sorting Application
- Implementation & Result
  - Multithreaded Sorting Application
    - 结果
  - Fork-Join Sorting Application
    - Quicksort
    - Mergesort
    - 结果
- Difficulties
- Reference

Abstract

- 编写了一个多线程排序程序，使用两个线程对整数列表的两个部分分别进行排序，第三个线程完成merge操作
- 使用Java编写了一个Fork-Join排序程序，利用`fork`函数和`join`函数实现快排和归并排序

Environment

- Ubuntu 18.04
- Linux 5.3.0-42-generic
- VMware Workstation Rro 15.5.0 build-14665864

Quick Start

编译

gcc编译

Multithreaded Sorting Application是c代码，直接使用如下gcc命令进行编译，注意因为使用了pthread库，所以此处需要加上`lpthread`参数

```
gcc proj3-1.c -lpthread -o proj3-1
```

jdk编译

Fork-Join Sorting Application是java代码，安装好jdk后直接使用如下命令进行编译。

```
javac Quicksort.java
javac Mergesort.java
```

测试代码

Multithreaded Sorting Application

```
./proj3-1 10 9 8 7 6 5 4 3 2 1 0
```

Fork-Join Sorting Application

```
java Quicksort 10 9 8 7 6 5 4 3 2 1 0
java Mergesort 10 9 8 7 6 5 4 3 2 1 0
```

Implementation & Result

Multithreaded Sorting Application

这部分代码用c进行编写，因为进程间共享全局变量，所以定义了全局变量`a`用于存储原数组，定义了全局变量`res`用于存储结果。

处理分为以下几步：

1. divide

divide部分在main函数中完成，直接将数组从中间分开后将起始位置传入子进程函数中即可。

2. sort

sort部分内使用冒泡排序，对两部分分别进行排序。

3. merge

merge部分在函数中对原数组前后两部分按照归并排序算法归并部分进行处理。

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int *a;
int *res;
int len;
int beg;
int mid;

void* sort(void *arg)
```

```
{
    int *s = arg;
    int end, temp;
    if (*s == mid)
        end = len;
    else
        end = mid;

    for (int i = *s; i < end; i++)
        for (int j = i + 1; j < end; j++)
            if (a[i] > a[j])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}

void* merge(void *arg)
{
    int s1 = 0;
    int s2 = mid;
    int pos = 0;
    while (s1 < mid && s2 < len)
    {
        if (a[s1] < a[s2])
        {
            res[pos] = a[s1];
            pos++;
            s1 ++;
        }
        else
        {
            res[pos] = a[s2];
            pos++;
            s2++;
        }
    }
    while (s1 < mid)
    {
        res[pos] = a[s1];
        pos++;
        s1++;
    }
    while (s2 < len)
    {
        res[pos] = a[s2];
        pos++;
        s2++;
    }
}

int main(int argc, char *argv[])
{
    pthread_t s1, s2, m;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
```

```
    len = argc - 1;
    a = (int *)malloc(len * sizeof(int));
    res = (int*) malloc(len * sizeof(int));

    for (int i = 0; i < len; i++)
        a[i] = atoi(argv[i + 1]);
    beg = 0;
    mid = len / 2;

    pthread_create(&s1, &attr, sort, &beg);
    pthread_create(&s2, &attr, sort, &mid);
    pthread_join(s1, NULL);
    pthread_join(s2, NULL);
    pthread_create(&m, &attr, merge, NULL);
    pthread_join(m, NULL);

    for (int i = 0; i < len; i++)
        printf("%d ", res[i]);
    printf("\n");

    free(a);
    free(res);
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/3/1$ gcc proj3-1.c -lpthread -o proj3-1
pan@pan-virtual-machine:~/桌面/osproj/3/1$ ./proj3-1 10 9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9 10
```

Fork-Join Sorting Application

Quicksort

快排部分的算法参考了数据结构中的算法实现，在此不做赘述，主要讲解如何使用java进行多线程programming。

c中的多线程实现主要是使用了`pthread_create`和`pthread_join`函数，而java中则使用了使用了`fork`和`join`函数。多线程的实现方式主要利用了以下代码分别对左右两部分在两个子进程中并行处理：

```
Quicksort left = new Quicksort(begin, low - 1, array);
Quicksort right = new Quicksort(low + 1, end, array);
left.fork();
right.fork();
left.join();
right.join();
```

完整代码如下：

```
import java.util.concurrent.*;

public class Quicksort extends RecursiveAction
{
    static final int THRESHOLD = 100;

    private int begin;
```

```
    private int end;
    private int[] array;

    public Quicksort(int begin, int end, int[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }
    protected void compute() {
        if (end - begin < THRESHOLD)
        {
            for (int i = 0; i < end; i++)
            {
                int temp = array[i];
                int flag = i;
                for (int j = i + 1; j < array.length; j++)
                {
                    if (array[j] < temp)
                    {
                        temp = array[j];
                        flag = j;
                    }
                }
                if (flag != i) {
                    array[flag] = array[i];
                    array[i] = temp;
                }
            }
        }
        else
        {
            int low = begin;
            int high = end;
            int k = array[begin];
            while (low != high)
            {
                while (low < high && array[high] >= k)
                    high--;
                if (low < high)
                {
                    array[low] = array[high];
                    low++;
                }
                while (low < high && array[low] <= k)
                    low++;
                if (low < high)
                {
                    array[high] = array[low];
                    high--;
                }
            }
            array[low]=k;

            Quicksort left = new Quicksort(begin, low - 1, array);
            Quicksort right = new Quicksort(low + 1, end, array);
            left.fork();
            right.fork();
            left.join();
```

```
            right.join();
        }
    }

    public static void main(String [] args){
        ForkJoinPool pool = new ForkJoinPool();
        int [] array = new int[args.length];

        for(int i = 0; i < args.length; i++){
            array[i] = Integer.parseInt(args[i]);
            System.out.print(array[i] + " ");
        }
        System.out.println("");

        Quicksort S = new Quicksort(0, args.length-1, array);

        pool.invoke(S);

        for(int i = 0; i < args.length; i++)
            System.out.print(array[i] + " ");
        System.out.println("");
    }
}
```

Mergesort

归并排序的代码基本与快速排序相同。多线程的实现方式主要利用了以下代码分别对左右两部分在两个子进程中并行处理，之后调用`merge`函数进行merge操作：

```
int mid = (begin + end) / 2;
Mergesort left = new Mergesort(begin, mid, array);
Mergesort right = new Mergesort(mid + 1, end, array);
left.fork();
right.fork();
left.join();
right.join();
merge(array, begin, mid + 1, end);
```

完整代码如下：

```
import java.util.concurrent.*;

public class Mergesort extends RecursiveAction
{
    static final int THRESHOLD = 100;

    private int begin;
    private int end;
    private int[] array;

    public Mergesort(int begin, int end, int[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }
```

```
    protected void merge(int [] array, int left, int mid, int right){
        int [] tmp = new int[end-begin+1];
        int i=left,j=mid,k=0;

        while (i < mid && j <= right)
        {
            if (array[i] < array[j])
                tmp[k++] = array[i++];
            else tmp[k++] = array[j++];
        }
        while (i < mid)
            tmp[k++] = array[i++];
        while (j <= right)
            tmp[k++] = array[j++];
        for (i = 0, k = left; k <= right;)
            array[k++] = tmp[i++];
    }

    protected void compute() {
        if (end - begin < THRESHOLD)
        {
            for (int i = 0; i < end; i++)
            {
                int temp = array[i];
                int flag = i;
                for (int j = i + 1; j < array.length; j++)
                {
                    if (array[j] < temp)
                    {
                        temp = array[j];
                        flag = j;
                    }
                }
                if (flag != i) {
                    array[flag] = array[i];
                    array[i] = temp;
                }
            }
        }
        else
        {
            int mid = (begin + end) / 2;
            Mergesort left = new Mergesort(begin, mid, array);
            Mergesort right = new Mergesort(mid + 1, end, array);
            left.fork();
            right.fork();
            left.join();
            right.join();
            merge(array, begin, mid + 1, end);
        }
    }

    public static void main(String [] args){
        ForkJoinPool pool = new ForkJoinPool();
        int [] array = new int[args.length];

        for(int i = 0; i < args.length; i++){
            array[i] = Integer.parseInt(args[i]);
```

```
            System.out.print(array[i] + " ");
        }
        System.out.println("");

        Mergesort S = new Mergesort(0, args.length-1, array);

        pool.invoke(S);

        for(int i = 0; i < args.length; i++)
            System.out.print(array[i] + " ");
        System.out.println("");
    }
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/3/2$ java Quicksort 10 9 8 7 6 5 4 3 2 1 0
10 9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9 10
pan@pan-virtual-machine:~/桌面/osproj/3/2$ java Mergesort 10 9 8 7 6 5 4 3 2 1 0
10 9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9 10
pan@pan-virtual-machine:~/桌面/osproj/3/2$
```

Difficulties

- Multithreaded Sorting Application部分编译时，最初没有加上`lpthread`参数，编译失败，查找资料后解决。
- Fork-Join Sorting Application部分要求使用java进行编写，这是我第一次独立编写java程序，所幸java和cpp风格类似，我通过几个样例程序进行了快速入门，初步掌握了java程序的编写。
- Fork-Join Sorting Application部分要求编译java代码，在安装环境时遇到了一些问题，通过查阅reference部分中提到的博客得到解决。

Reference

- Operating System Concept 10^{th} edition
- Source code for the 10th edition of Operating System Concepts https://github.com/greggagne/osc10e
- linux安装jdk环境 https://blog.csdn.net/lyhkmm/article/details/79524712